http://ovh.to/CTQWgzP

Electronique numérique 3 : Introduction

Patrick.Garda@upmc.fr

Objectifs & Organisation

- Conceptuels
  - Conception circuits électroniques numériques en VHDL
  - Machines à Etat
  - Microarchitecture
  - Composants : FPGA
- Opératoires
  - Logiciel simulation et synthèse
  - Prototypage sur carte FPGA ALTERA
- Faire le lien
  - NUM1, NUM2, ARCHI, CODESIGN

- Enseignement
  - 5 x 2h cours
  - 5 x 4h TP
- Contrôle des connaissances
  - Moyenne 4x1/2h CC
  - Moyenne TP
- Ressources pour l'UE
  - Sakai ?

Electronique numérique 3 : micro-architecture d'un processeur mono-cycle

Patrick.Garda@upmc.fr

Introduction

- ## Objectif
  - Proposer une méthode de conception de fonctions électroniques numériques

- ## Moyen
  - Exemple d'un cœur de processeur

- ## Source
  - Prof. James L. Johnson
  - Computer Science Dept
  - Western Washington University
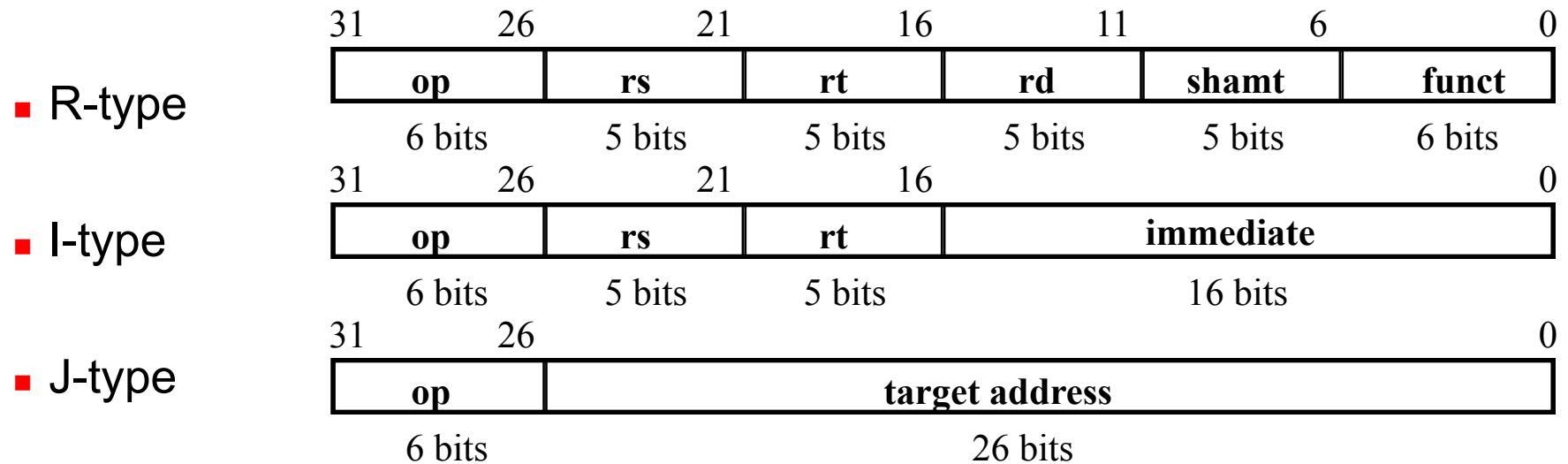 - http://faculty.cs.wwu.edu/johnson/

How to Design a Processor: step-by-step

1. Analyze instruction set => datapath <u>requirements</u>
   a. the meaning of each instruction is given by the *register transfers*
   b. datapath must include storage element for ISA registers
      and possibly some intermediate results
   c. datapath must support each register transfer

2. Select set of datapath components and establish clocking methodology

3. Assemble datapath meeting the requirements

4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.

5. Assemble the control logic

The MIPS Instruction Formats

- All MIPS instructions are 32 bits long. The three instruction formats:

- R-type

31 26	21	16	11	6	0
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- I-type

31 26	21	16	0
op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

- J-type

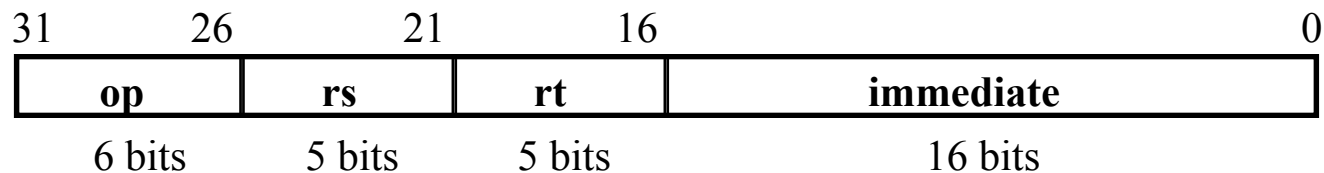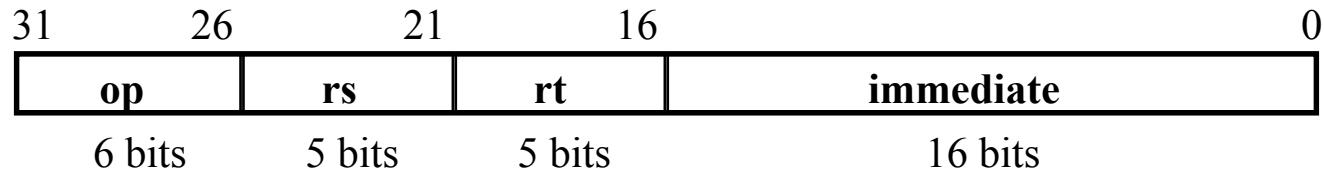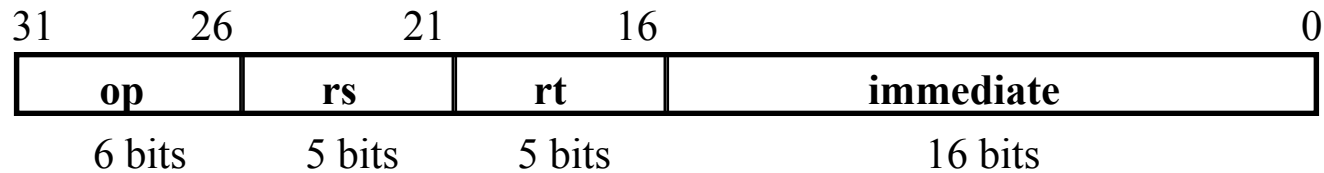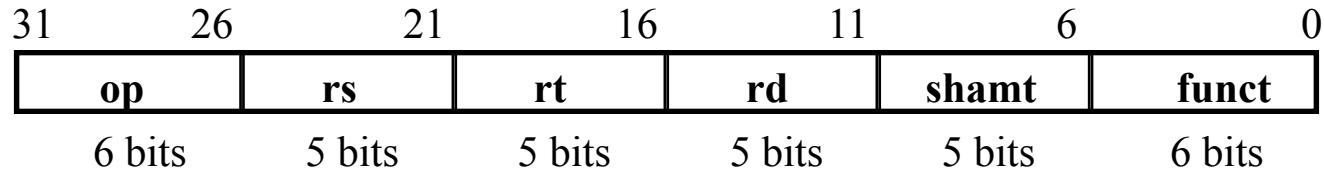| 31 26	0
op	target address
6 bits	26 bits

- The different fields are:

  - op: operation

  - rs, rt, rd: source and destination register

  - shamt: shift amount

  - funct: selects operation variant

  - address / immediate: address offset or immediate value

  - target address: target address of the jump instruction

Step 1a: The MIPS-lite Subset

- add, sub, and, or, slt; e.g.
    - add rd, rs, rt
    - sub rd, rs, rt
- immediate; e.g.
    - ori rt, rs, imm16
- memory reference; e.g.
    - lw rt, rs, imm16
    - sw rt, rs, imm16
- branch:
    - beq rs, rt, imm16

31	26	21	16	11	6	0

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

31	26	21	16	0

op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

31	26	21	16	0

op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

31	26	21	16	0

op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

Instruction definition

- RTL gives the <u>meaning</u> of the instructions
- All start by fetching the instruction

op | rs | rt | rd | shamt | funct = MEM[PC]

op | rs | rt | Imm16   = MEM[PC]

inst	Register Transfers	
ADD	R[rd] <– R[rs] + R[rt];	PC <– PC + 4
SUB	R[rd] <– R[rs] – R[rt];	PC <– PC + 4
ORI	R[rt] <– R[rs]	zero_ext(Imm16);	PC <– PC + 4
LOAD	R[rt] <– MEM[R[rs] + sign_ext(Imm16)];	PC <– PC + 4
STORE	MEM[R[rs] + sign_ext(Imm16)] <– R[rt];	PC <– PC + 4

BEQ		if (R[rs] == R[rt])
		    PC <- PC + 4 + (SignExt(imm16) x 4)
		else
		    PC <– PC + 4

I – Chemin de données datapath

Patrick.Garda@upmc.fr

Step 1b: ISA storage elements

- Memory
    - instruction & data
- Registers (32 x 32)
    - read RS
    - read RT
    - Write RT or RD
- PC
- SR

Step 1c: ISA operations

- Extender
- Add and Sub register or extended immediate
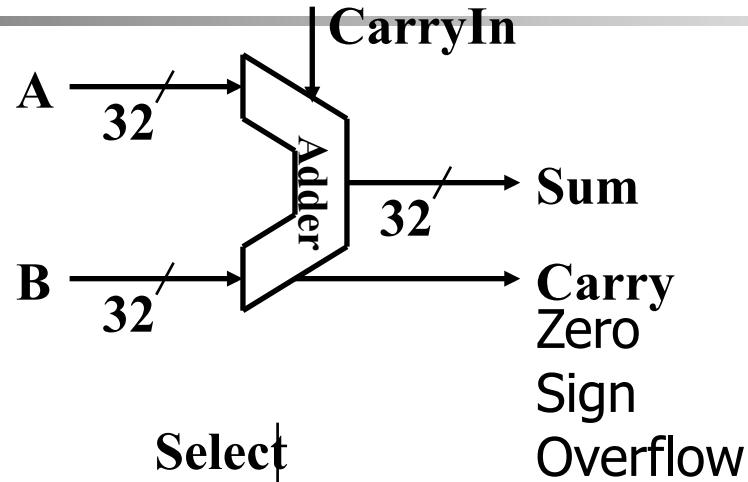- Add 4 or extended immediate to PC
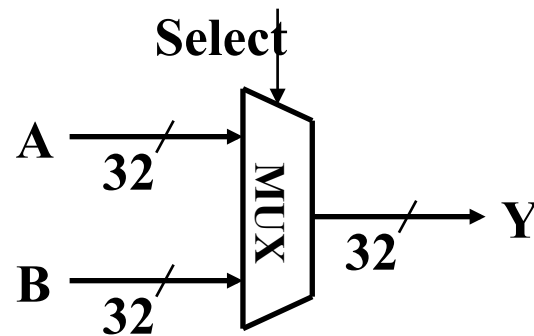- OR Bitwise

Step 2: Components of the Datapath

- Combinational Elements

- Storage Elements
  - Clocking methodology
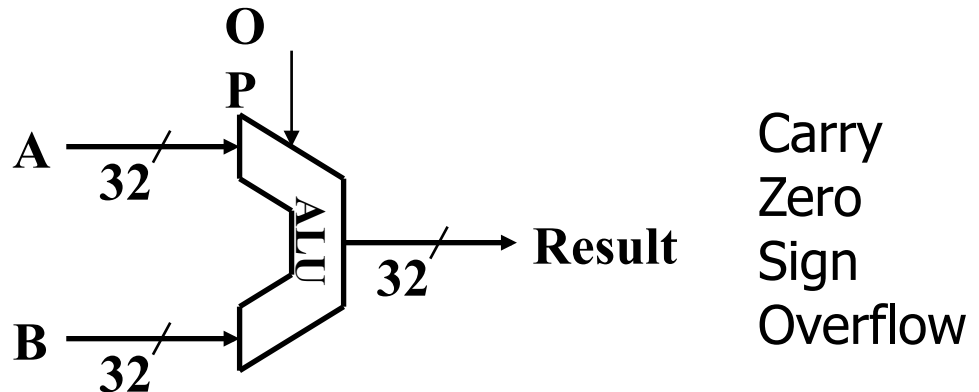
Combinational Logic Elements

- Adder

A —/32→ [Adder] CarryIn
B —/32→ [Adder] —/32→ Sum
Carry
Zero
Sign
Overflow

- MUX

Select
A —/32→ [MUX]
B —/32→ [MUX] —/32→ Y

- ALU

OP
A —/32→ [ALU]
B —/32→ [ALU] —/32→ Result

Carry
Zero
Sign
Overflow

Storage Element: Register

- Register

  - Similar to the D Flip Flop except
    - N-bit input and output
    - Write Enable input

  - Write Enable:
    - negated (0): Data Out will not change
    - asserted (1): Data Out will become Data In

Write Enable

Data In → **Data Out**

N   N

Clk
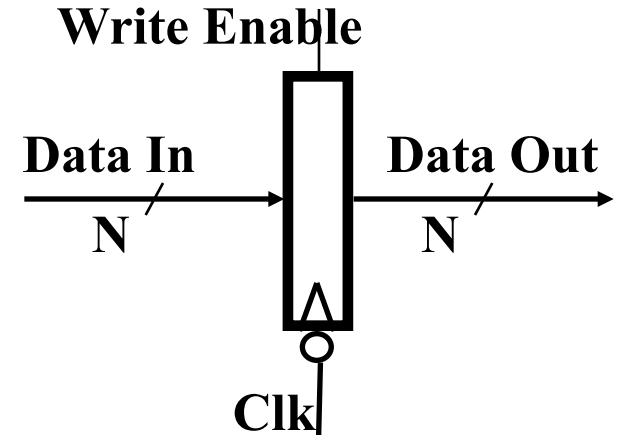
Storage Element: Register File

- Register File consists of 32 registers:
  - Two 32-bit output busses:
    busA and busB
  - One 32-bit input bus: busW

- Register is selected by:
  - RA (number) selects the register to put on busA (data)
  - RB (number) selects the register to put on busB (data)
  - RW (number) selects the register to be written from busW (data) when Write Enable is 1

- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - RA or RB valid => busA or busB valid after "access time."

RW RA RB

Write Enable 5 5 5

busW 32

Clk

32 32-bit Registers

busA 32

busB 32
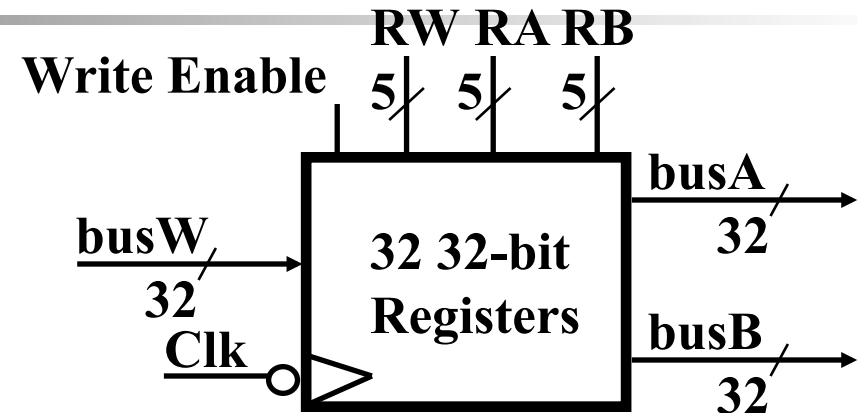
Storage Element: Idealized Memory

Write Enable   **Address**

Data In → [ ] 32 → **DataOut** → 32

**Clk**

- Memory (idealized)
  - One input bus: Data In
  - One output bus: Data Out

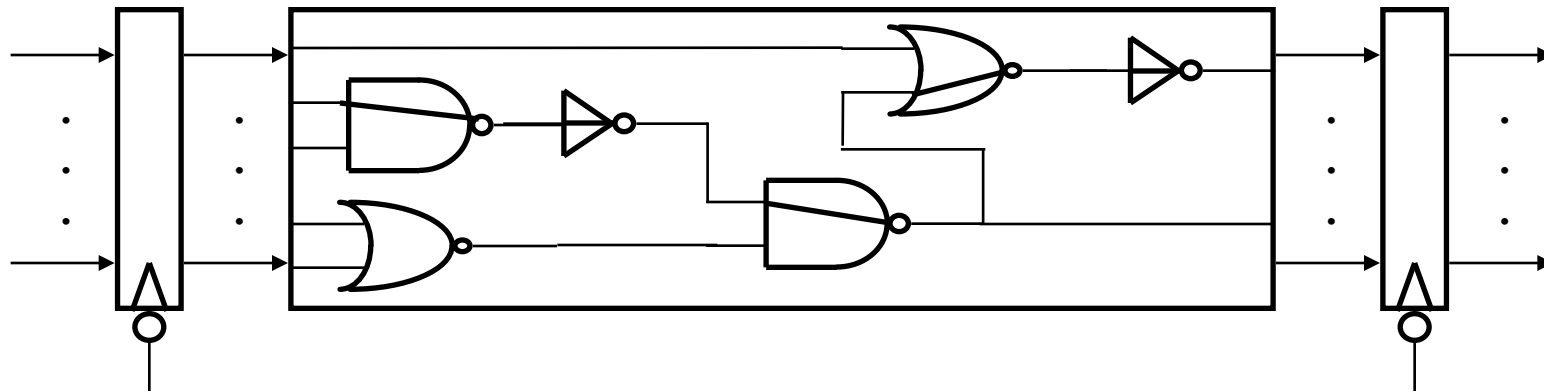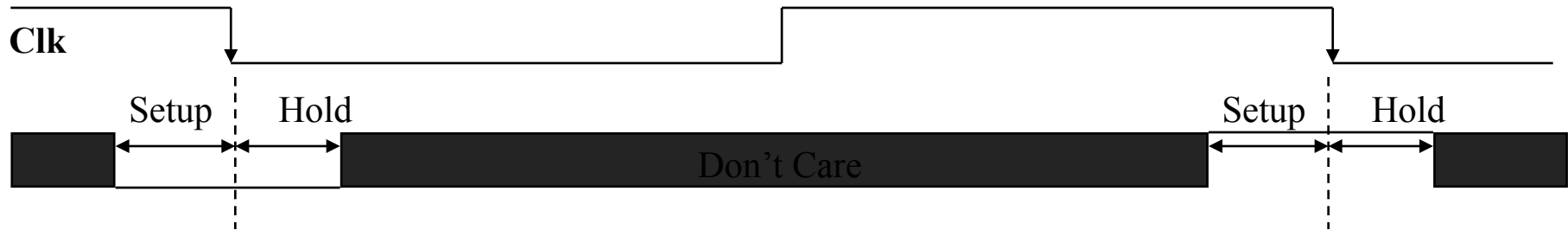- Memory word is selected by:
  - Address selects the word to put on Data Out
  - Write Enable = 1: address selects the memory word to be written via the Data In bus

- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
  - Address valid => Data Out valid after "access time."
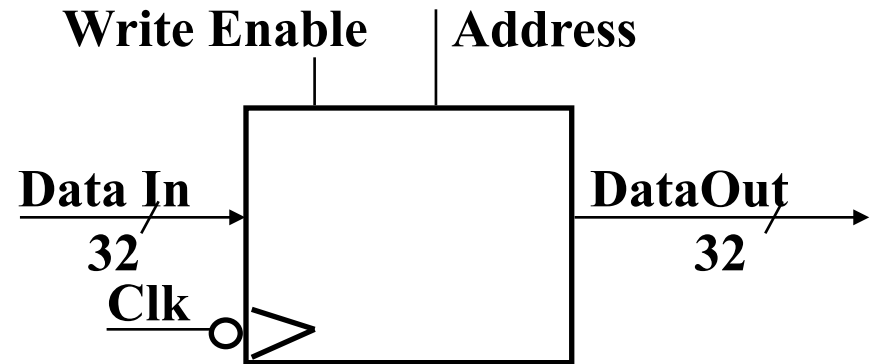
Clocking Methodology

- All storage elements are clocked by the same clock edge

- Cycle Time = CLK-to-Q + Longest Delay Path + Setup + Clock Skew
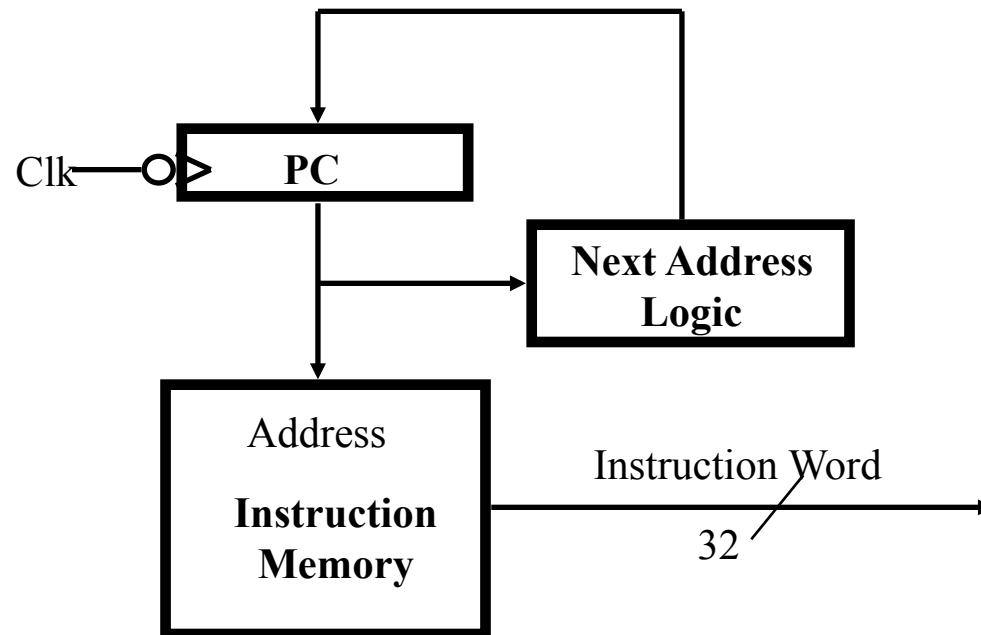
- (CLK-to-Q + Shortest Delay Path - Clock Skew) > Hold Time

Step 3

- Register Transfer Requirements –> Datapath Assembly
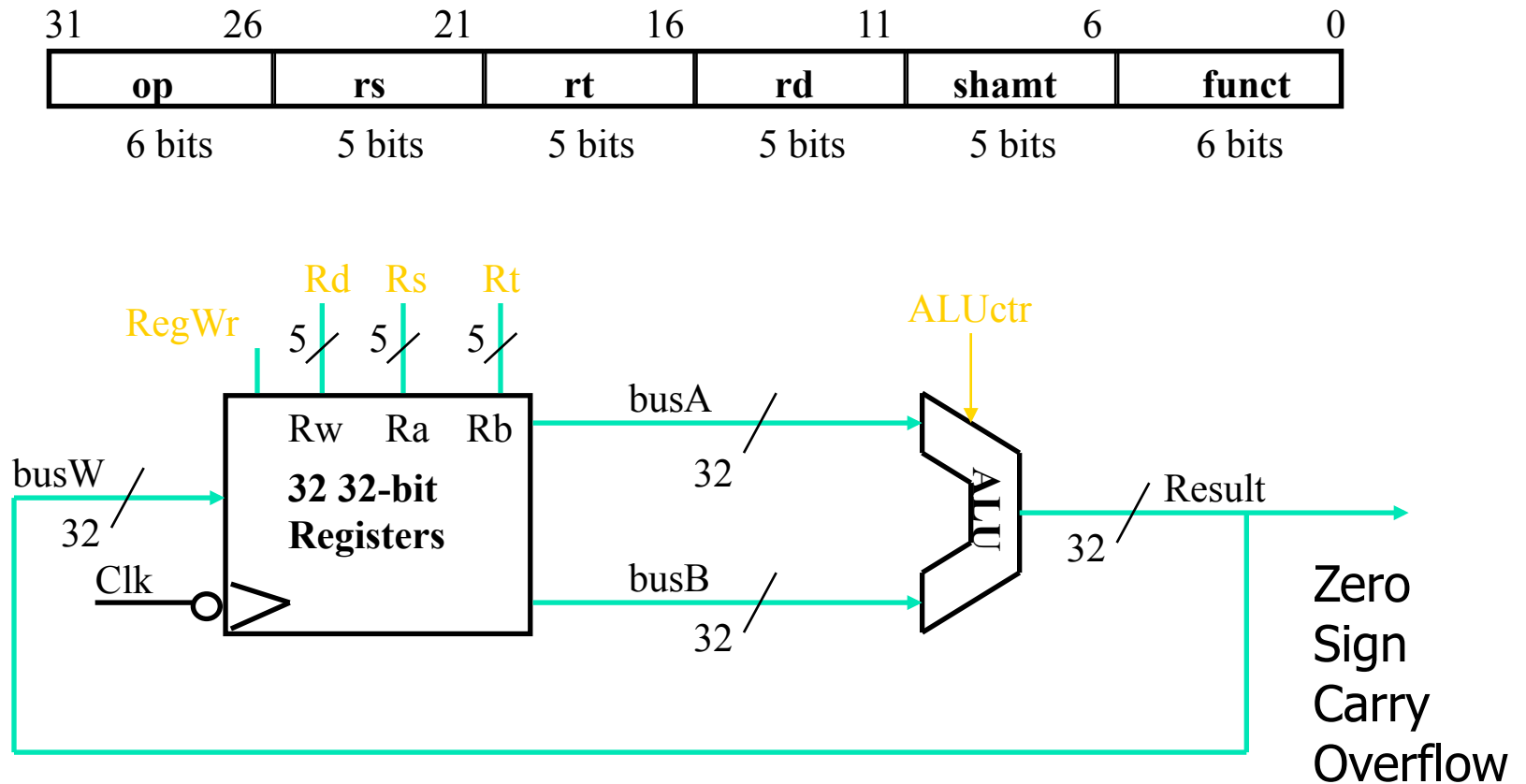
- Instruction Fetch

- Read Operands and Execute Operation

- The common RTL operations
  - Fetch the Instruction: mem[PC]
  - Update the program counter:
    - Sequential Code: PC <- PC + 4
    - Branch and Jump: PC <- "something else"

```
Clk ──o▷   PC

                        Next Address
                           Logic

            Address
                              Instruction Word
          Instruction                  32
           Memory
```

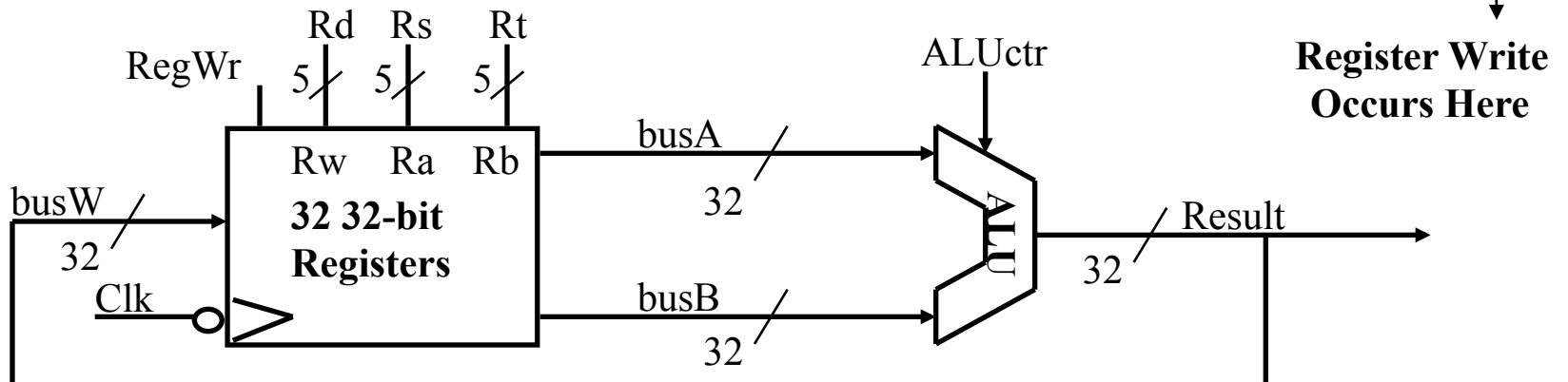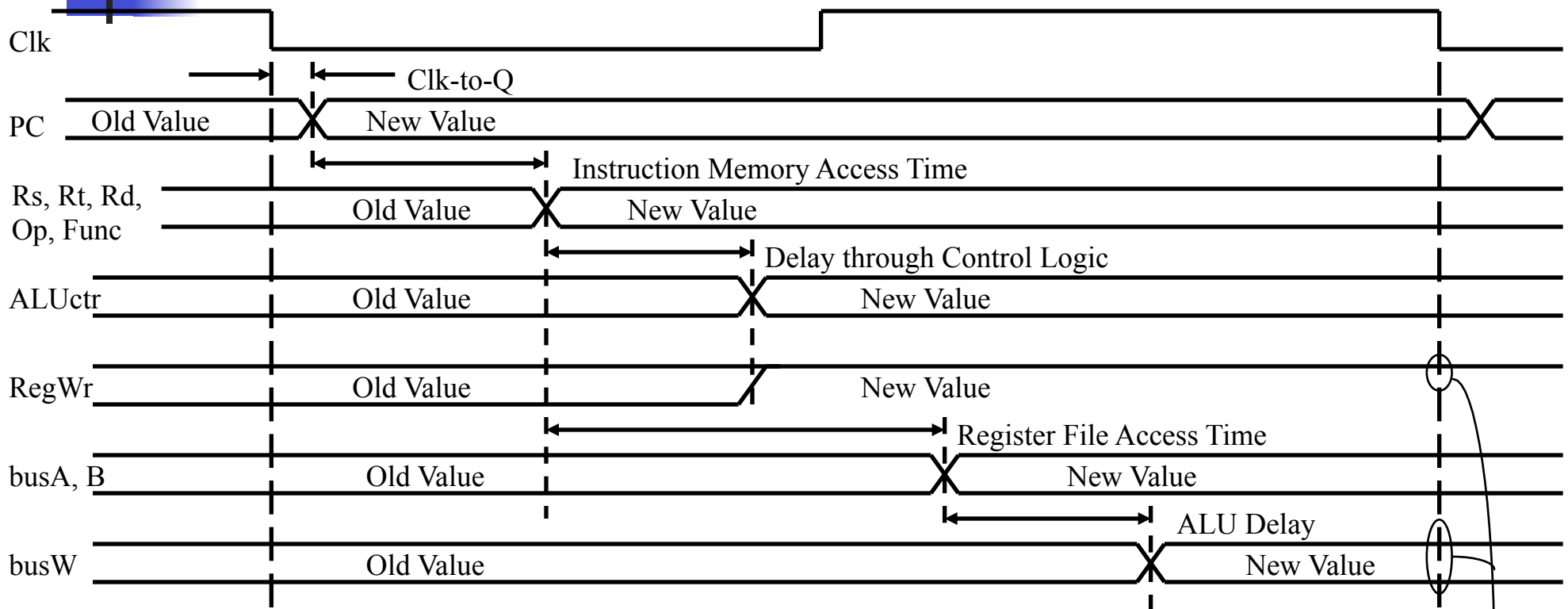# 3b: Add & Subtract

- R[rd] <- R[rs] op R[rt]   Example: add   rd, rs, rt
  - Ra, Rb, and Rw come from instruction's rs, rt, and rd fields
  - ALUctr and RegWr: control logic after decoding the instruction
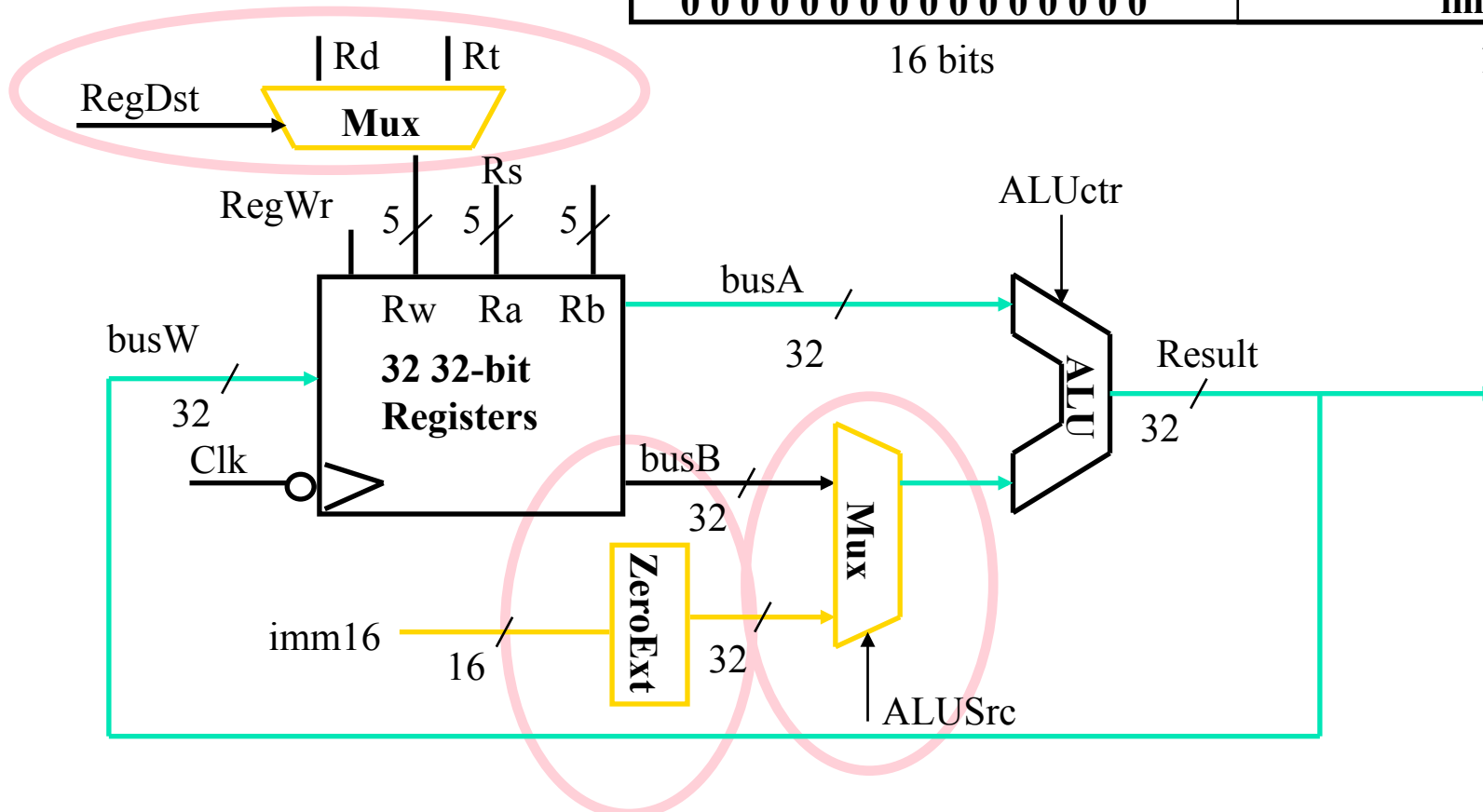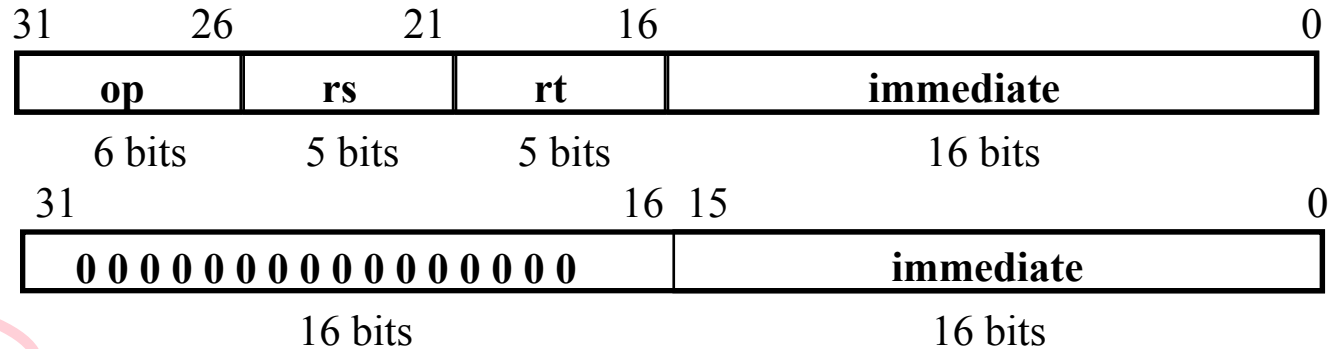
31	26	21	16	11	6	0
op	**rs**	**rt**	**rd**	**shamt**	**funct**	
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

Zero
Sign
Carry
Overflow

Register-Register Timing

Clk

Clk-to-Q

PC   Old Value   New Value

Instruction Memory Access Time

Rs, Rt, Rd,
Op, Func   Old Value   New Value

Delay through Control Logic

ALUctr   Old Value   New Value

RegWr   Old Value   New Value

Register File Access Time

busA, B   Old Value   New Value

ALU Delay

busW   Old Value   New Value

Register Write Occurs Here

RegWr   Rd   Rs   Rt
5   5   5

ALUctr

busW
32

Rw   Ra   Rb
32 32-bit Registers

Clk

busA
32

busB
32

ALU

Result
32

3c: Logical Operations with Immediate

- R[rt] <- R[rs] ORI ZeroExt[imm16]]

31	26	21	16	0
op	rs	rt	immediate	
6 bits	5 bits	5 bits	16 bits	

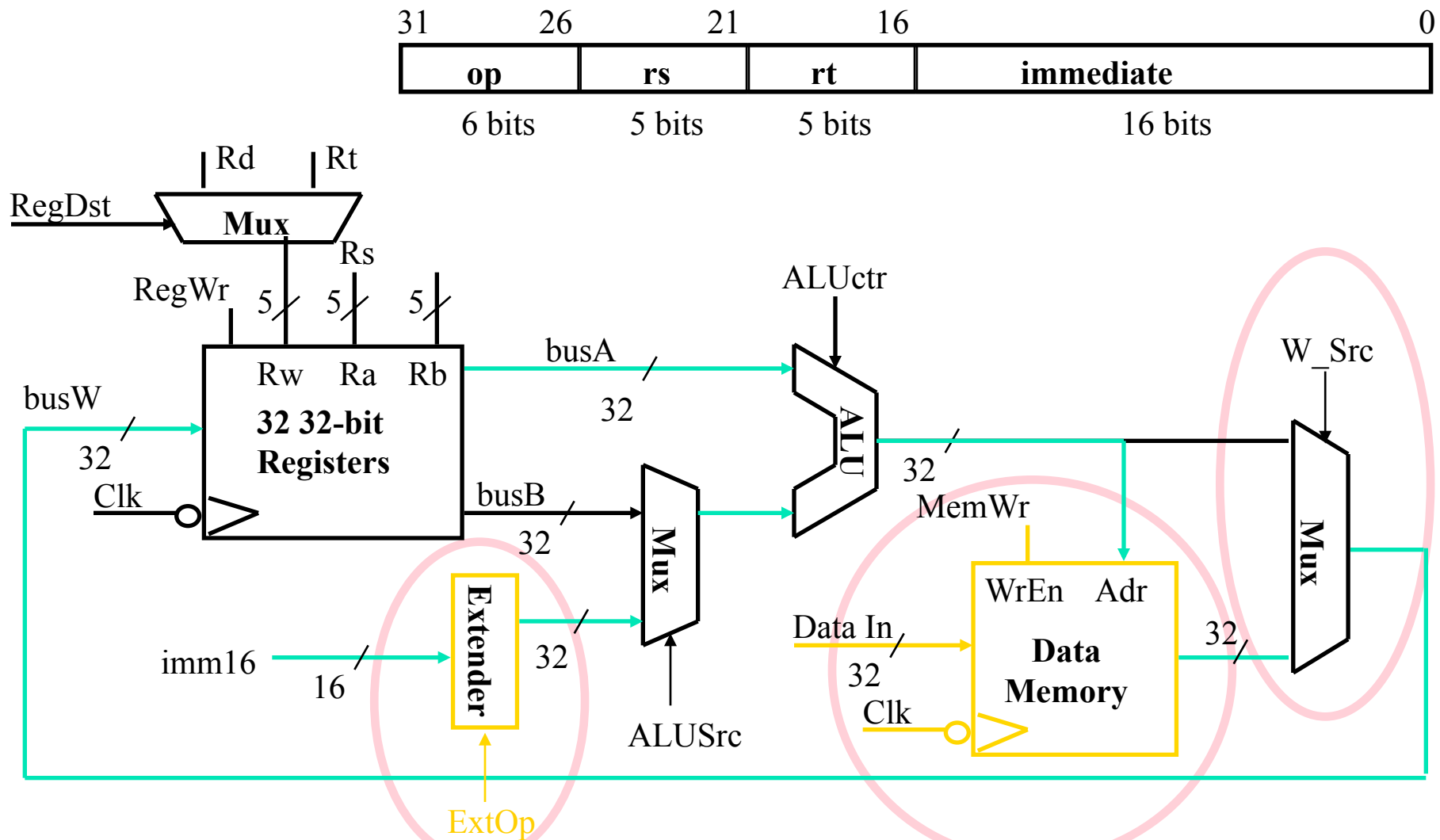| 31	16	15	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		immediate	
16 bits		16 bits	

C2

- $- b_{15} 2^{15} + b_{14} 2^{14} + \ldots$

- $= (-2 + 1) b_{15} 2^{15} + b_{14} 2^{14} + \ldots$

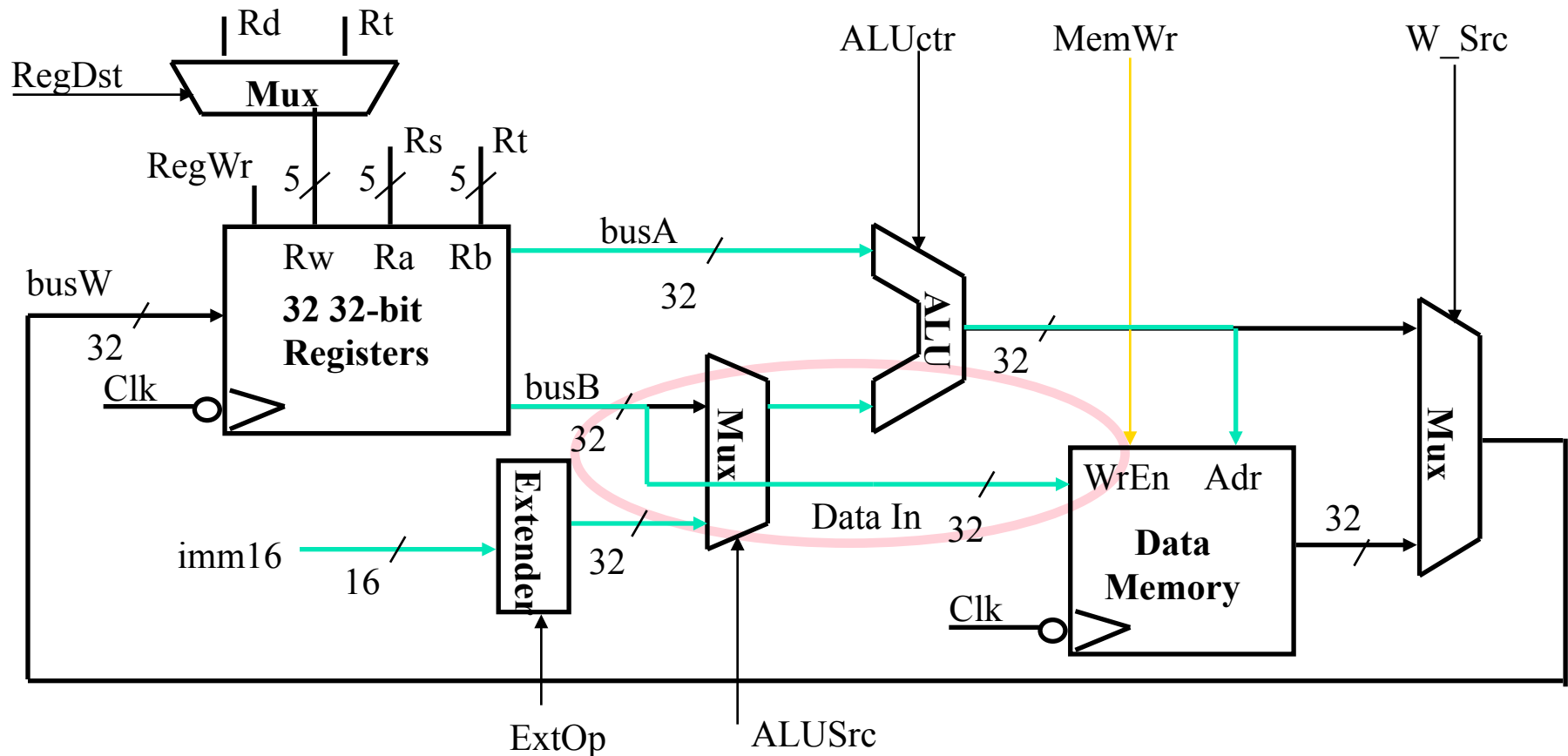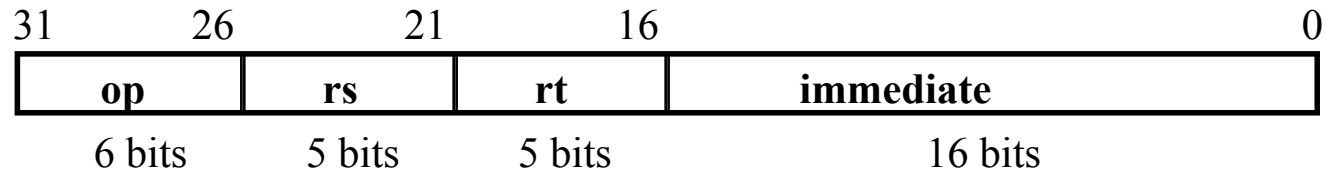- $= - b_{15} 2^{16} + b_{15} 2^{15} + b_{14} 2^{14} + \ldots$

3d: Load Operations

- R[rt] <- Mem[R[rs] + SignExt[imm16]]

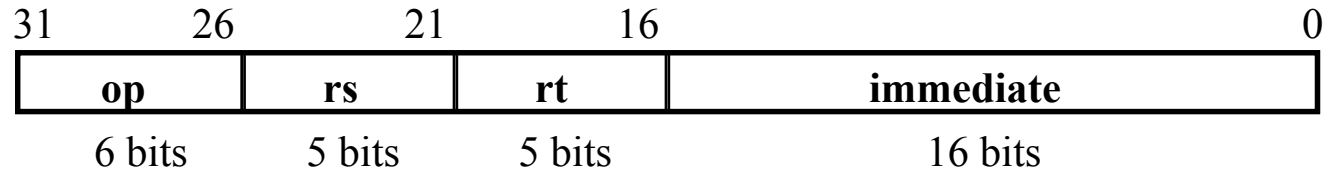| 31	26	21	16	0
op	rs	rt	immediate	
6 bits	5 bits	5 bits	16 bits	

3e: Store Operations

- Mem[R[rs] + SignExt[imm16]] <- R[rt]

3f: The Branch Instruction

31	26	21	16	0
op	rs	rt	immediate	
6 bits	5 bits	5 bits	16 bits	

- beq   rs, rt, imm16

  - mem[PC]   Fetch the instruction from memory

  - Equal <- R[rs] == R[rt]   Calculate the branch condition
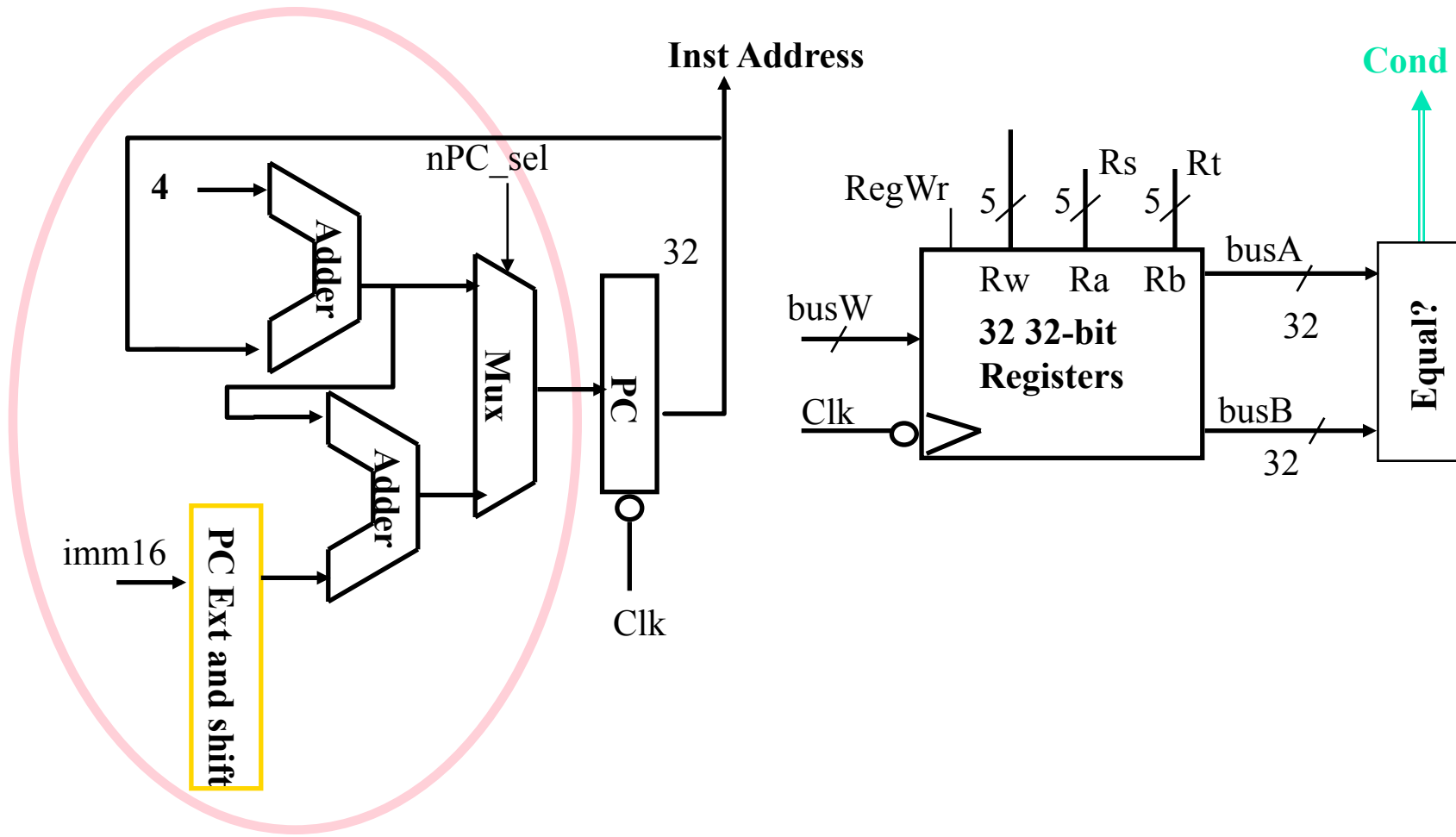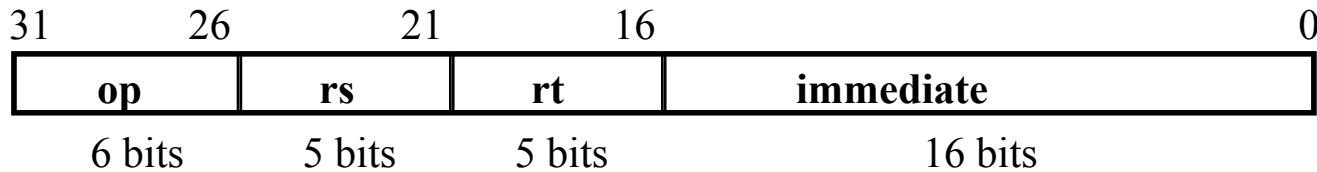
  - if (COND eq 0)   Calculate the next instruction's address
    PC <- PC + 4 + (SignExt(imm16) x 4)
  - else
    PC <- PC + 4

Datapath for Branch Operations

- beq rs, rt, imm16

Datapath generates condition (equal)

31	26	21	16	0
op	rs	rt	immediate	
6 bits	5 bits	5 bits	16 bits	

Putting it All Together: A Single Cycle Datapath